Popular Computing

The world's only magazine devoted to the art of computing.

January 1979 Volume 7 Number 1

6555959		62956919
	2	
6555961		62956921
	10	
6555971		62956931
	32	
6556003		62956963
	10	
6556013		62956973
	6	
6556019		62956979
	8	
6556027		62956987
	4	
6556031		62956991
	6	
6556037		62956997
	12	
6556049		62957009
	12	
6556061		62957021
	2	
6556063		62957023
	16	
6556079		62957039

Patterns in Primes

Contest Result

Patterns in Primes

Our 17th contest appeared in our issue number 65, July 1978. The following strings of consecutive prime numbers were given:

5651	1146791
5653	1146793
5657	1146797
5659	1146799
5669	1146809
5683	1146823
5689	1146829
5693	1146833
5701	1146841

Both of these strings have the same pattern of differences; namely, 2, 4, 2, 10, 14, 6, 4, 8. The contest called for the longest strings of consecutive prime numbers having the same difference pattern.

The winning result--actually, two of them--comes from Sam Wagner, Bluffton, Ohio, shown here and on the cover of this issue. We believe that it is unlikely that longer lists will ever be found.

Publisher: **Audrey Gruenberger**
Editor: **Fred Gruenberger**
Associate Editors: **David Babcock**
Irwin Greenwald
Patrick Hall
Contributing Editors: **Richard Andree**
William C. McGee
Thomas R. Parkin
Edward Ryan
Art Director: **John G. Scott**
Business Manager: **Ben Moore**

POPULAR COMPUTING is published monthly at Box 272, Calabasas, California 91302. Subscription rate in the United States is $20.50 per year, or $17.50 if remittance accompanies the order. For Canada and Mexico, add $1.50 per year. For all other countries, add $3.50 per year. Back issues $2.50 each.

21 713 957		42 248 597
	2	
21 713 959		42 248 599
	12	
21 713 971		42 248 611
	6	
21 713 977		42 248 617
	6	
21 713 983		42 248 623
	30	
21 714 013		42 248 653
	10	
21 714 023		42 248 663
	6	
21 714 029		42 248 669
	2	
21 714 031		42 248 671
	10	
21 714 041		42 248 681
	12	
21 714 053		42 248 693
	6	
21 714 059		42 248 699
	8	
21 714 067		42 248 707

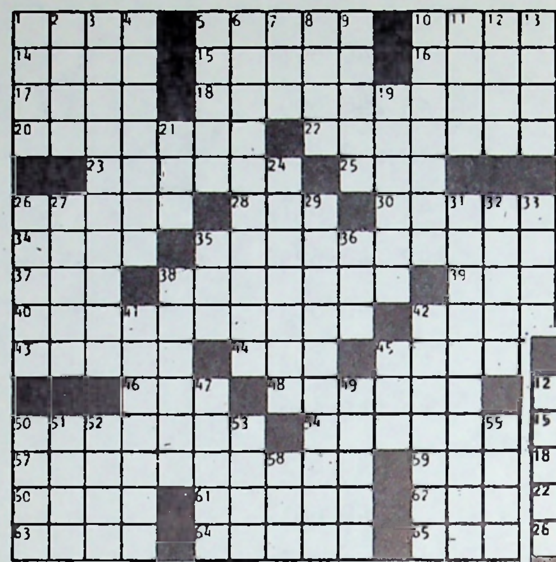
Mr. Wagner calculated all the differences between successive values of the first 6,016,001 primes, and assumed that each of these differences could be contained in 15 bits.

[Actually, up to the 6,000,000th prime, the largest such difference is 220, which is nicely contained in only 8 bits.]

The computing was done on a Data General Eclipse system with a 96 megabyte disk, using Fortran.

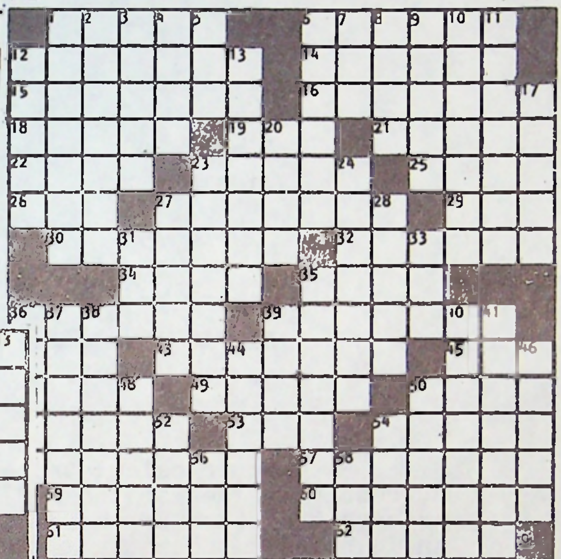
All the differences except one are even numbers, so it is expedient to store the half differences. Differences over 15 were changed to zero, so that strings of twelve half-differences could be expressed in 48 bits for ease of manipulation. Successive sifting reduced the file of differences to eligible strings that finally yielded the two sequences shown. Mr. Wagner is richer by a TI-58 programmable calculator, and we are richer by an interesting piece of knowledge.

PC70--4

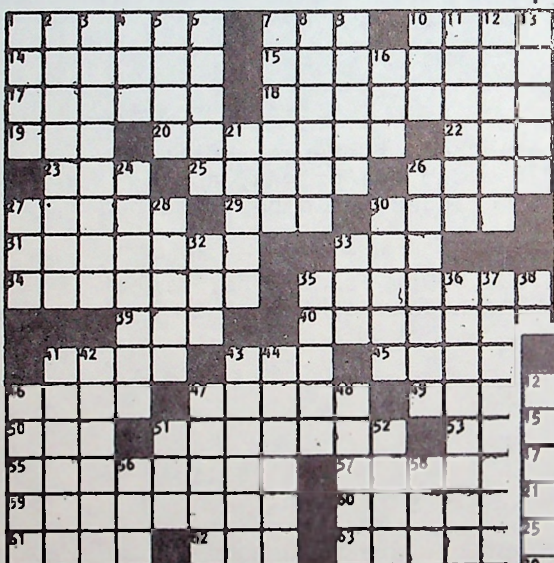


P

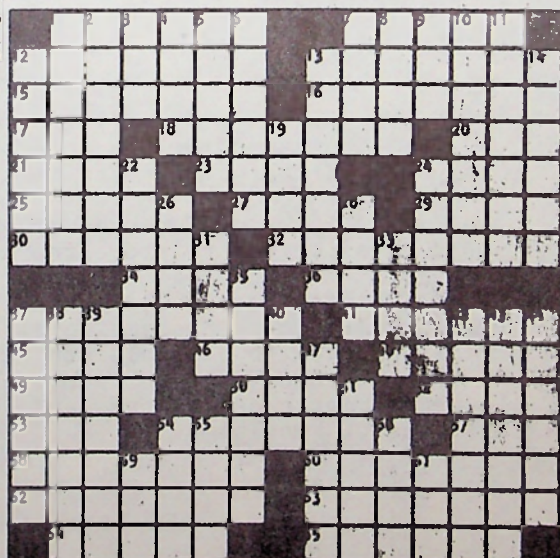
T



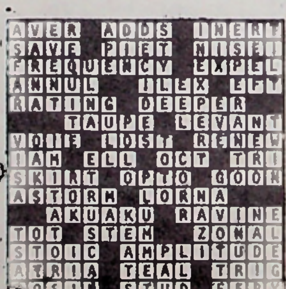
Q



R



S



U

homework 7

The daily paper carries crossword puzzles like the ones shown here. Each puzzle consists of a 15 x 15 grid of squares, with some of the 225 squares blacked out. The number of blacked out squares varies from 28 to 42 and averages 34. In the last decade there have been over 3000 of these puzzles published, and it is rare to have any pattern of the black squares repeat. It does happen, however.

Picture yourself as having those 3000 puzzles at hand. We want to find all those puzzles having the same pattern (two or more times), including rotations and mirror images.

Your problem is this: Is this a computer problem?

If you conclude that it is, then outline the steps in your proposed computer solution. If you conclude that it is not, then outline your alternative method. The game we are playing is this: we do want to solve the problem (that is, don't waste time debating the usefulness of the project) and we do want to minimize our work, the costs, and/or the elapsed time to a solution--the question is, what is the best way to go about it?

We can list some attacks on this problem that won't work:

(a) We can't call the newspaper to ask them. The daily puzzles are syndicated and arrive by mail at the newspaper office. Moreover, that wasn't the problem; we want to find the duplicates, not just know that they're there.

(b) If we could track down the person who composes the puzzles, it is doubtful if he or she would know, either. Why should he?

(c) No, there is no optical scanner for sale somewhere that will read black squares on a puzzle and put that information into machine-readable form. Our primary entry medium to the available computer is still the punched card.

The Crossword Puzzle ~~homework~~ Problem

The problems that have appeared in our Homework series have been used in computing classes; they are time-tested written assignments. The Crossword Puzzle Problem on the preceeding pages is different--the "homework" is intended to be time spent on thinking over the economics of a solution. As it turns out, it would be foolish to use a computer on the problem, but it usually lures beginning students into devising elaborate and complex computer solutions. The problem can provoke an hour or more of healthy discussion, which is its intent.

Most beginners will plunge in with something like: "We could compare the elements in the first row of one of the patterns..." only to be dismayed at the thought that before any comparing can be done, the data to be compared must somehow be entered into the machine, a small point that they have overlooked.

So if a computer is to be used, the critical problem concerns data representation and data entry, and such things as the cost of keystroking and how keystroking can be validated.

Regarding the cost of keystroking: in any large city, you can buy about 9000 keystrokes per hour, for about $4.50, with assurance of no more than two errors. (A competent keypunch operator will make many mistakes each hour, but will recognize that fact and correct each error. No more than two errors will survive per hour.) If you would like to have your work correct, then you will have it keyed twice and have the two decks of cards compared by machine. You are now getting effectively 10 keystrokes per penny (with the probability of 4 errors per 81,000,000).

Students will invent elaborate schemes for digitizing the patterns of black squares, in order to present alphameric data to the keypunchers. The simplest such scheme is to punch zeros for the blank squares and ones for the black squares, and use up 225 keystrokes for each pattern. Each pattern will thus consume three complete IBM cards, with 225 columns of data and 5 columns on each card to identify the pattern and the card number (we learned a long time ago never to deal with unidentified data in our business). For the 3000 puzzles, this figures out to 18000 fully punched cards, costing about $720 for the punching and about $40 for the cards themselves.

Eventually some alert student finds a shortcut to all this. Why punch 225 characters per puzzle? The patterns all exhibit symmetry of some sort, so that each pattern is uniquely defined by half of its black squares, or perhaps even by one-quarter of them.

And now a class will begin to zero in on an efficient solution to the problem. Of the six patterns given with the assignment, no two are alike. How long did it take you to verify that? Patterns T, Q, and S all have the upper left square blacked out, but a glance at the next blacked out square of each pattern quickly eliminates any possibility of duplication. Similarly, patterns P and U have black squares starting in the 5th column, but the string of black squares is 3 long for pattern P and only 2 long for pattern U.

Given the 3000 puzzles, it would take perhaps an hour to arrange them in no more than 8 stacks, depending on the position of the first black square. Each of these stacks can then be similarly sorted into substacks depending on the position of the second black square, and so on. All the duplicates in the 3000 puzzles can be located in this way in two to three hours, by one person, by hand.

For an actual collection of 3000 such puzzles, the following is the state of affairs by actual count:

One case of 6 alike.

One case of 5 alike.

Two cases of 4 alike.

Six cases of 3 alike.

Thirty cases of 2 alike.

Eight cases of mirror-images.

2887 unique patterns.

BOOK REVIEW

BASIC: An Introduction To Computer Programming

by Robert J. Bent and George C. Sethares
Brooks/Cole 1978, 8 1/2 x 11 softcover, $9.95

> ...stresses the techniques and programming principles useful in solving problems (in any language)....each chapter addresses a new problem-solving process. The progression of topics is related to semantics, that is, what the language can do, not according to syntax....flowcharting is introduced just after the IF statement, since any introduction before this would be seemingly contrived...By page 19 of the text the student can write and run a computer program!!!
>
> ---from the brochure's hyperbole

The title of this book places it in a category of textbooks which does not suffer from overcrowding; namely, the set of "language" books that also attempt to teach the rudiments of computing.

What are the aspects of such a book that could be done badly? Let's list some of them in priority order, worst first:

1. The "facts" about the language involved are wrong.

2. Various essential aspects of the language are omitted.

3. The book fosters poor computing practices.

4. Programs given in the book as models are wrong.

5. Programs given in the book as models are poorly done (e.g., spaghetti programs).

6. Examples and exercises are poorly chosen.

7. The writing is bad.

The above list, although not comprehensive, can serve to calibrate a new textbook. Let's apply it, with some charity, to the Bent and Sethares text.

An early exercise in the book (page 47) and its accompanying flowchart (Figure A) is given here:

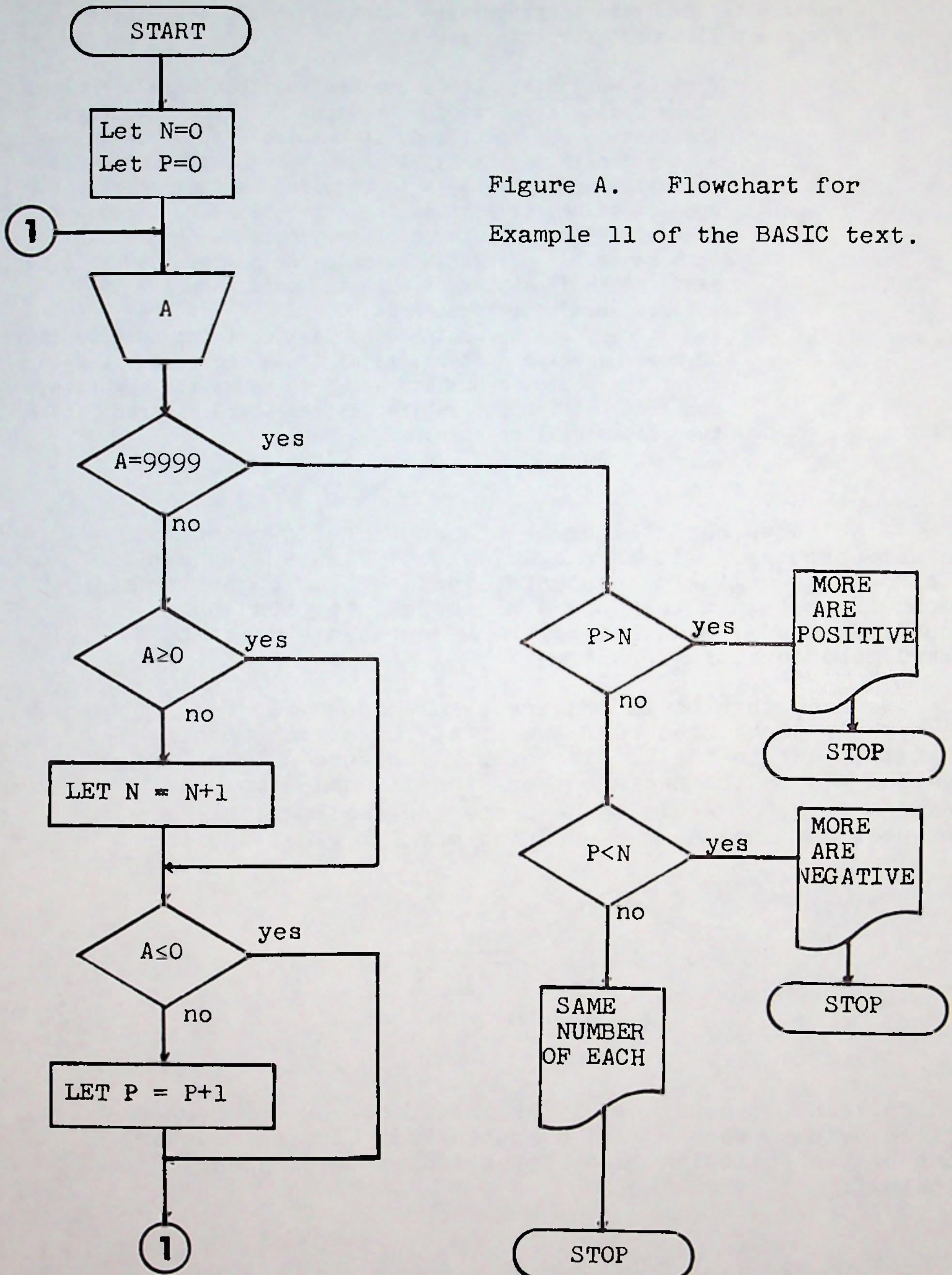


Figure A. Flowchart for Example 11 of the BASIC text.

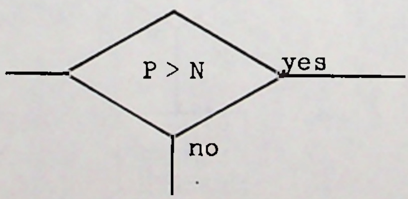
Example 11. A list of numbers is to be typed at the terminal to determine whether there are more positive numbers or more negative numbers in the list. Appropriate messages should be printed. Prepare a flowchart for this task.

Problem analysis: Let's use the variable name A for the number being typed at the terminal. Since the length of the list is not specified, it must be assumed that lists of any length may be typed in. One way to allow for variable-length lists is to require the user to type a special value for A after the entire list has been entered. Of course, the computer must be programmed to recognize this number. For this example the number 9999 will be used and an IF statement with the condition A = 9999 will be used to recognize it.

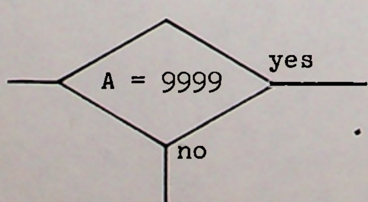
To determine whether the list contains more positive or more negative numbers, we will use the variable P to count the positive numbers and N to count the negative numbers. After the entire list has been entered, these two values will be compared.

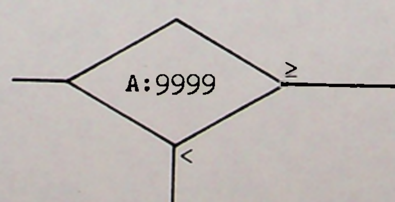
Also shown here (Figure B) is another flowchart for the same problem. It is suggested that Figure A, shown as a model to beginning students, fosters poor computing (among other things that one might object to, its logic flounders around) and that Figure B represents a start toward good computing practices.

We have tons of literature (whole books, in fact) on the symbols to be used on flowcharts, but almost nothing on what to put in the little boxes. Figure A is a fine example. Does the decision box--the diamond--imply a question mark, or would it be better to use English sentences and include the question mark? That is, is



good English, or not? Would an insistence on proper English help to foster good computing? For that matter, which of the following is a better indicator of clear thinking?:





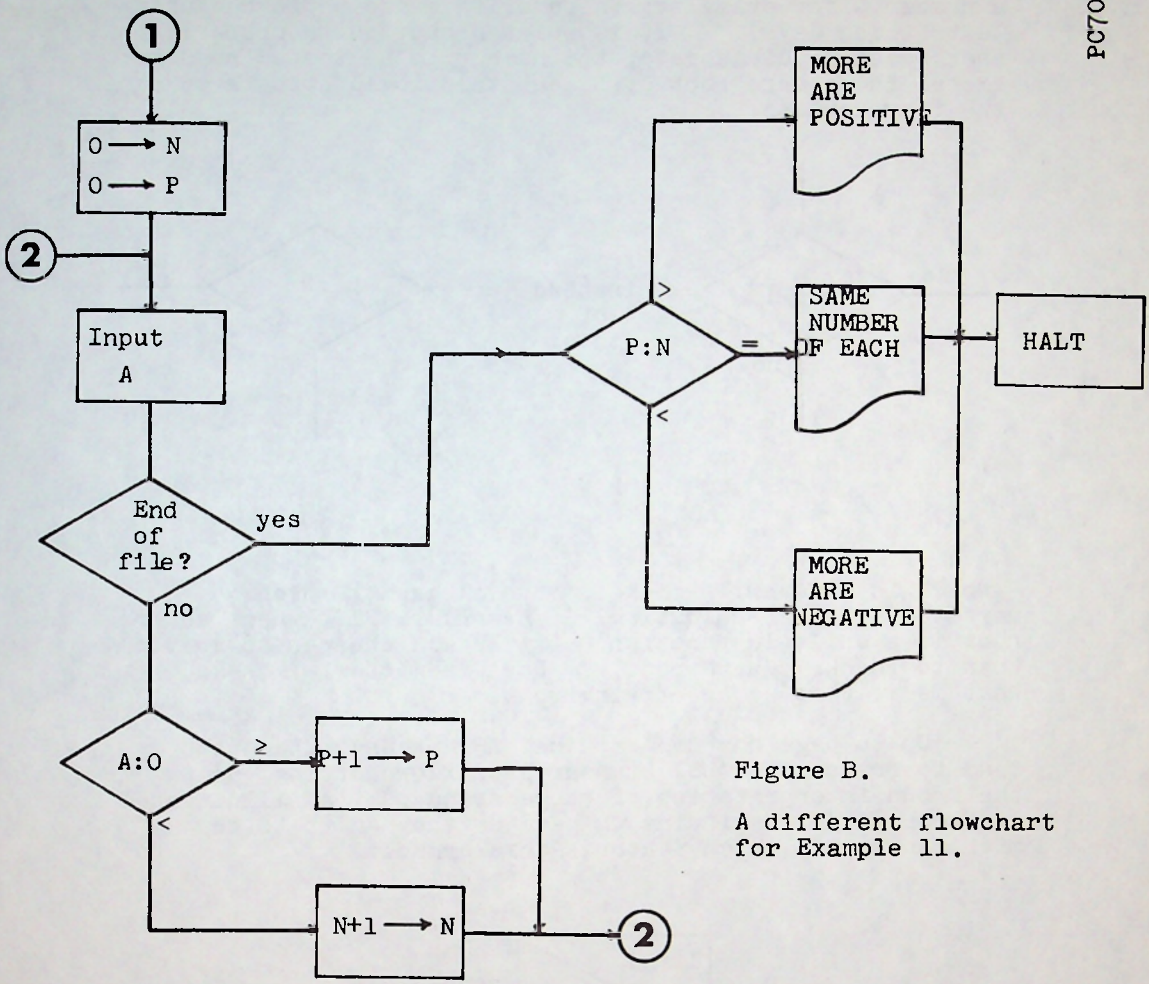


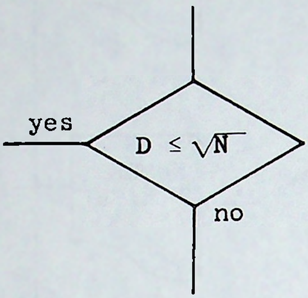
Figure B.

A different flowchart for Example 11.

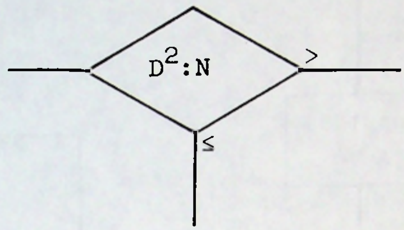
What will the student learn, when he sees a model flowchart in which A is compared to zero twice (needlessly) and P is compared to N twice (needlessly)? The logic in Figure B is at least as clear, is much shorter, and is arranged so that parallel ideas appear in parallel. The logic of Figure A is the sort of thing that might be developed in class on the blackboard, but the end result shown in a textbook should do better than that.

The whole case for or against this book could rest on the fine points between Figures A and B, but we can look further.

On page 61, the reader is shown the logic of testing a number N for primality (a favorite exercise in beginning texts these days). It is pointed out, quite properly, that possible divisors of the number to be tested need not exceed the square root of N, but this logic appears on the flowchart as:

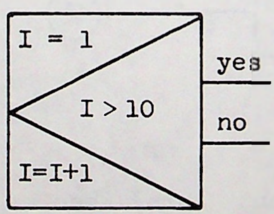


instead of this:



--which is clearer?; which is more elegant?; which represents better computing? Even in BASIC, where square root is a built-in function, it is still cheaper to multiply than to invoke square root. Does it matter? Well, what is it that we are trying to teach?

Up to page 88 (the book has 239 pages), the authors tend to follow the ANSI standards of flowcharting, using their own interpretation of those standards, as illustrated by Figure A. Beginning on page 88 they shift to the Michigan style, which features this symbol:



We have precious few agreed-upon standards in computing, but one that we do have is a standard set of flowchart symbols. If, given such a standard, everyone goes his own way, we tend toward chaos. It is irrelevant to argue that some non-standard notation is better, or that it represents how things "should be." It took 22 years just to get agreement that a flowchart decision should be a diamond. It is now more than 30 years in which we have not agreed as to whether to slash the letter or the digit. Beginning students (to whom this book is aimed) are not served by the use of parochial and oddball notation.

In the chapter on sorting, the authors begin with bubble sorting--but it is the version of bubble sorting in which K(K+1)/2 comparisons, and possible interchanges, will be made regardless of the initial ordering of the K items being sorted. Again, they are fostering less than good computing. Much can be learned from a thorough discussion of the various versions of bubble sorting (particularly the version that capitalizes on any ordering that already exists in the data).

All the above criticisms might be debatable; there are points of view involved, and fundamental differences in beliefs about what constitutes good computing and how to teach it. What really triggered a lengthy review of a rather prosaic (but, on the whole, well-done) text was the following.

In the chapter on random numbers, there are the following exercises:

> 5. Simulate tossing three coins ten times. The output should be a list of terms such as HHH, HTH, HHT, and so on.

> 10. Generate a list A of 100 different numbers. Any such list will do. For example, A(1)=1, A(2)=2, and so on. Write programs to do the following.
>
> a. Create a list B of approximately 20 different numbers selected randomly from list A.
>
> b. Randomly select one number from list A, and then randomly select another from the remaining 99.
>
> c. Randomly select exactly 20 different numbers from list A.

> 16. Two knights begin at diagonally opposite corners of a chessboard and travel randomly about the board but always making legitimate knight moves. Calculate the number of moves before one knight captures the other. However you number the squares, each knight's move should be printed as it is taken.

Problem 5 is a traditional finger exercise, to get the student started using the new function. Problem 10 is somewhat more challenging, and by the time a student gets to part (c), he will welcome eagerly any suggestions from an experienced teacher as to an efficient way to go about the assigned task.

But then we have problem 16, which is in a different league entirely. We will discuss this problem at some length. The point to be made is that the three exercises are several orders of magnitude apart in difficulty, and the student should be informed of this (even high school algebra books mark the difficult problems with a star) and be given some help to get started.

One of the first problems to be solved in the problem of the two knights is how to represent, in numbers, the board position. Figure C shows a possible scheme, with markers in the starting position for the normal size of chess board (8 x 8). At this point, we are already planning ahead to a generalized version of the problem, with the board size varying from 4 x 4 up to 14 x 14.

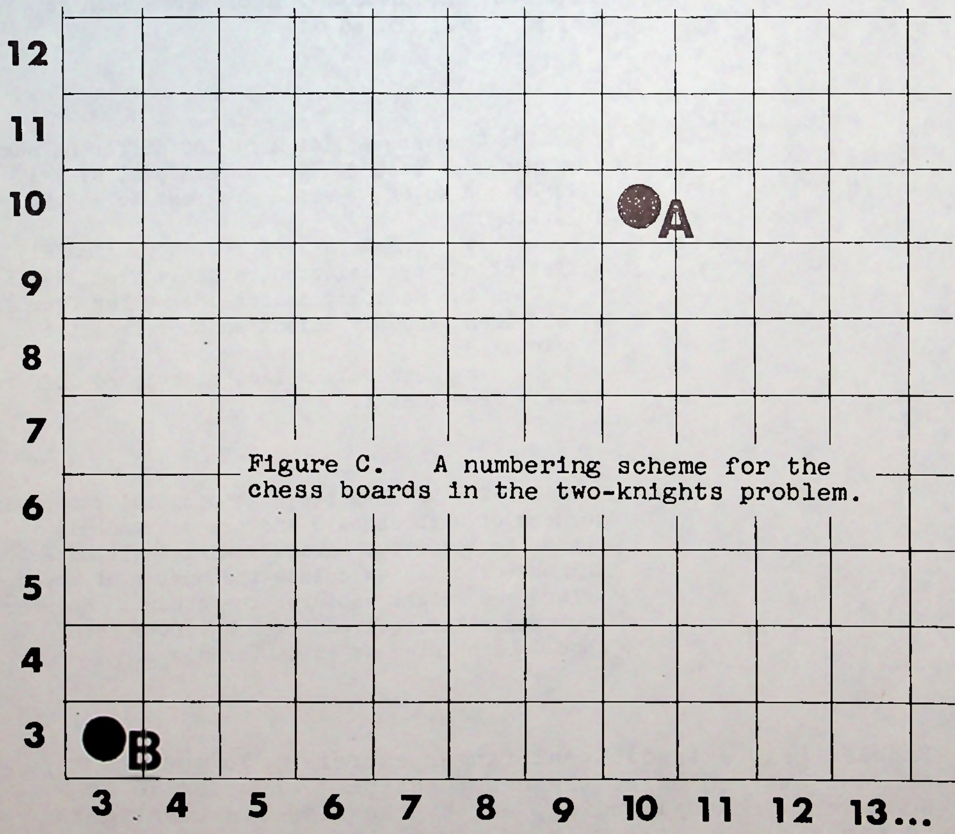


Figure C. A numbering scheme for the chess boards in the two-knights problem.

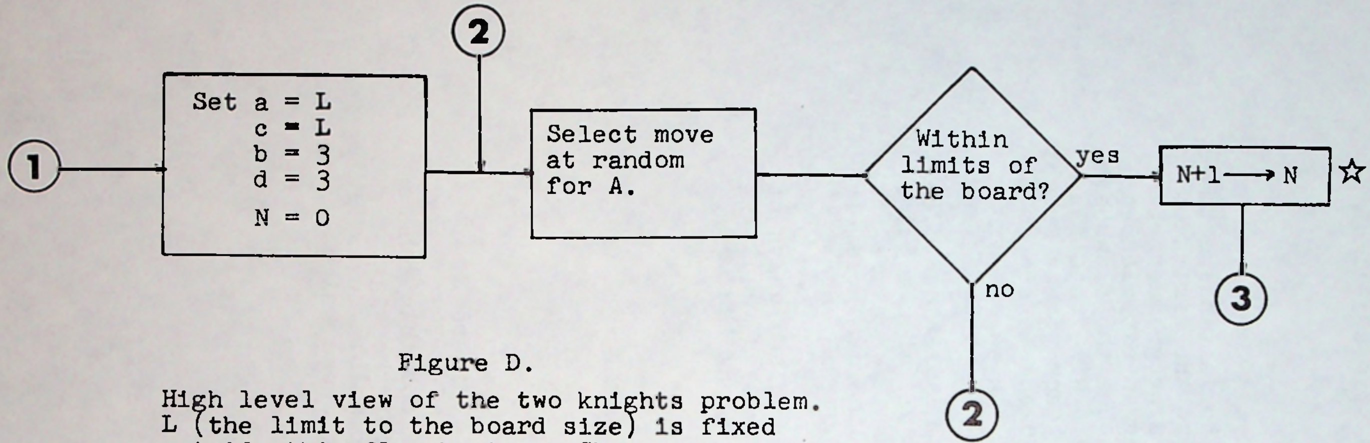


Figure D.

High level view of the two knights problem. L (the limit to the board size) is fixed outside this flowchart. Chess board is numbered in both dimensions from 3 to L.

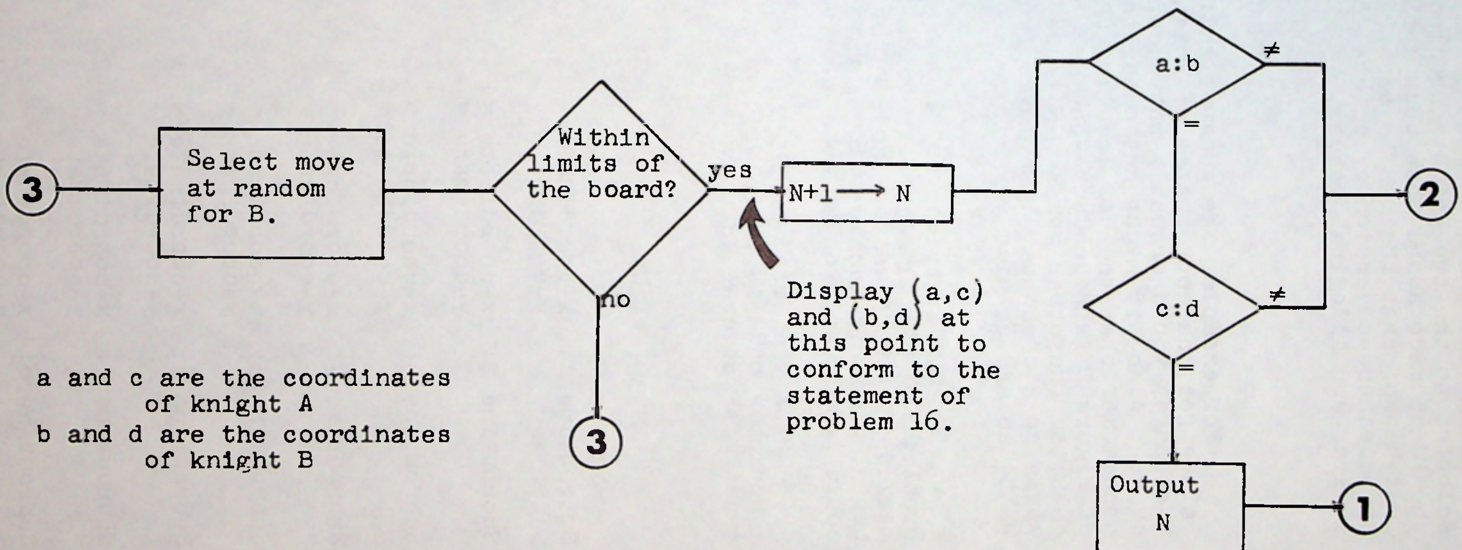


Figure D shows a broad-brush flowchart of a possible solution to the two-knights problem. A beginning student should be made aware that such a flowchart does not emerge full-blown; that the one he sees (all pretty and neat) is the cleaned-up version of several false starts, blind alleys, and illogical messes.

For example, the generalization (to initialize the values for the coordinates of knight A to the limits of the board) would probably not occur to a programmer right away. Also, it might not occur to even an experienced programmer that the box labelled (*) on the the flowchart of Figure D is redundant--the same result will obtain by changing the other box to $N + 2 \longrightarrow N$ (each time a knight moves on a chessboard, he moves from a black square to a white square or vice versa).

The first attempt at flowcharting a solution to this problem had all the equality tests inserted at the point marked (*), thus wasting gobs of machine time on useless tests, until it became evident that a coincident position could result only on an even move. Thus we learn.

From the broad-brush view to the nuts and bolts of a workable code is the next big step (probably requiring the drawing of several low-level flowcharts along the way).

Subroutine 1:	x-coordinate = 2	
	y-coordinate = -1	
Subroutine 2:	x-coordinate = 1	
	y-coordinate = -2	
Subroutine 3:	x-coordinate = -2	
	y-coordinate = -1	
Subroutine 4:	x-coordinate = -1	
	y-coordinate = -2	
Subroutine 5:	x-coordinate = 2	
	y-coordinate = 1	
Subroutine 6:	x-coordinate = 1	
	y-coordinate = 2	
Subroutine 7:	x-coordinate = -2	
	y-coordinate = 1	
Subroutine 8:	x-coordinate = -1	
	y-coordinate = 2	

Figure E.

A possible scheme for the eight possible moves of a chess knight.

Board size	Number of cases	Average number of moves	Standard deviation
4	63	10.22	13.17
5	53	36.30	32.97
6	54	65.44	59.11
7	74	47.00	37.62
8	35	68.06	65.52
9	74	108.57	95.91
10	55	122.84	100.34
11	75	125.84	112.51
12	35	144.17	112.32
13	54	219.81	193.80
14	25	294.16	128.36

Some results for the two-knight problem.

Figure F.

For example, the logical operation at Reference 2 is "Select move at random for knight A." This could be done in many ways. One possible way is the following. Set up 8 subroutines as in Figure E. Use whatever random number generator you have (e.g., the RND function that is built into your BASIC, although that might be the worst choice) to select a random integer between 1 and 8; name this integer R. Then call subroutine R. The values of x and y thus produced can be applied to (a,c) or (b,d) in order to effect one of the 8 possible legal moves that a chess knight can make.

For the 8 x 8 board, using the numbering scheme of Figure C, the result of applying a legal move to either knight A or knight B must meet the conditions that the coordinates a, b, c, or d must be less than or equal to 10 and greater than or equal to 3, or else the "legal" move has taken us off the board and a different move must be selected, again at random.

Figure F shows some results for various sized chess boards. The result for the 7 x 7 board seems questionable, but at least we have an answer for Problem 16 of the book. It is possible that there is a fundamental difference in the random action of the two knights, depending on whether the board has an odd number of cells on a side or an even number of cells--thus, as always, the end result of this tiny bit of research is an indication that more research is needed.

Lo! the poor Indian!

...Alexander Pope

There is a legend that says that Manhattan island was purchased from the Indians in 1627 for $24. If that $24 had been invested at 3%, compounded annually, how much would it amount to in 1978?

769267.01903352597916745758082995569741738273679358079789898 3
5040973693683217414348941606983612384291104875433815898074 15
5194768879544915332447300256623072539549606165661183475468 98
8982695786754881065846473318865733706806461174362059712800 77
5142107996919439876974850816386145344217454866883217038454 94
4264768533011696610203221178690092021087542269189731491237 06
9576799387748650274660831578012968364440300920507270181057 33
4573613440954696809480533272395023906997759859939289801842 48
8456255412476598046766701513194007695142415905064607826673 93
2151012513852362460174131289657571922854556282389087746139 67
5368522694870702546037592807509485696243504001750259653821 61
8009475452635724722591310690362067638152960635 28

The figures above are the result, as calculated step-by-step, retaining every digit at every step. On the facing page are more results for this problem. All these numbers are answers to the same well-defined problem, and they are all correct.

We are conditioned to the belief that there is always just one answer to a well-defined problem (that is, after the first shock of discovering that our answer of 53/3 to the problem in our arithmetic book didn't quite agree with the back of the book, where it was given as 17 2/3, or maybe as 17.67). So how can there be ten <u>different</u> answers, all correct, to the same problem?

The long calculation above can be checked, somewhat. The number of digits (708) checks with logarithms. The low-order digit cycles 4, 2, 6, 8, 4, 2, 6, 8 through the solution; the low-order two digits have a cycle of 20; the low-order three digits have a cycle of 100. Thus, some small calculation can verify that the 528 at the end of the number is correct.

Calculated in fixed point arithmetic, with each year's result rounded to the nearest cent.	769509.07
Calculated in fixed point arithmetic, with each year's result truncated to the nearest cent.	763936.33
Calculated in fixed point arithmetic, with each year's result truncated to the nearest 1/10 cent.	768734.195
Calculated in floating point arithmetic, step by step for 351 years, using a 9-significant digit package.	769267.03
Calculated in floating point arithmetic, step by step for 351 years, using a 12-significant digit package.	769267.02
Using logarithms and the compound interest formula, 12 significant digit logarithms.	769267.02
Using logarithms and the compound interest formula, 6 significant digit logarithms.	769266.00
Using logarithms and the compound interest formula, 5 significant digit logarithms.	769050.00
Using logarithms, and the compound interest formula, 4 significant digit logarithms.	780500.00

More results for the Manhattan Problem (1978)

($24 at 3% for 351 years, compounded annually)

Incidentally, if the Indians could have commanded a more modern 10%, instead of 3%, their bank account today would be about 8000 times the current annual Gross National Product of the United States.

TODAY IS TUESDAY, MARCH 83, 1907.

BOMBER WING SPAN IS 258 MILLIMETERS.

ISSUE OVERTIME CHECK FOR $3,872,654.24

7 X 8 = 63.

3375400264 X 165751322 = 5358979323846264338327955.

DIAGNOSTIC PASSED OK--THE COMPUTER WORKS PERFECTLY.

DIAMETER OF CONTROL RODS SHOULD BE 31 FEET.

VEHICLE 283 MADE 78 TRIPS MAY 19, 1978.

SOCIAL SECURITY NUMBER IS 85361429.

GASOLINE CONSUMPTION WAS .000318 MILES PER GALLON.

40,001,385 IS A PRIME NUMBER.

YOUR DIET REQUIRES 12.8 LB. OF BUTTER PER DAY.

RIVER TEMPERATURE IS 512.8 DEGREES F.

HOME PHONE NUMBER IS 34078236.

(Everything above is a reproduction of actual computer printout)

GIGO

Output Is A Function Of Input